A Way-Halting Cache for Low-Energy High-Performance Systems

Chuanjun Zhang*, Frank Vahid**, Jun Yang, and Walid Najjar
*Department of Electrical Engineering, Department of Computer Science and Engineering
University of California, Riverside
{chzhang, vahid, junyang,najjar@cs.ucr.edu}
**Also with the Center for Embedded Computer Systems, UC Irvine

Abstract: ***We have designed a low power four-way set-associative cache that stores the four lowest-order bits of all way's tags into a fully associative memory, which we call the halt tag array. The comparison of the halt tag array with the desired tag occurs concurrently with the address decoding that determines which tag and data ways to read from. The halt tag array pre-determines most tags that cannot match due to their low-order four bits mismatching. Further accesses to ways with known mismatching tags are then halted, thus saving power. Our halt tag array has the additional feature of using static logic only, rather than dynamic logic used in highly-associative caches, making our cache consumes even less power. Our result shows 55% savings of memory access related energy over a conventional four-way set-associative cache. We show nearly 2x energy savings compared with highly associative caches, while imposing no performance overhead and only 2% cache area overhead.***

I. INTRODUCTION

Caches may consume nearly 50% of a microprocessor's power [9][12]. Cache designers, for both high-end and embedded processors, must compromise between performance, cost, size, and power/energy dissipation.

In this paper, we introduce a new cache design, which we call a *way-halting* cache that reduces the cache's internal activities to access nearly the ideal minimum tag and data ways without any performance overhead – neither in the cache access time nor in the hit rate. Our cache is four-way set-associative, though the method can be applied to an arbitrary set associativity. We divide each of the four tag arrays into two sub-arrays: the first sub-array (the halt tag array) holds only the low-order 4 bits of each tag and the other sub-array (the main tag array) holds the remaining bits of each tag. A way-halting cache checks *all* the (four-bit) tags in the halt tag array in parallel with address index decoding, in contrast to traditional approaches that only check the tags in the cache set specified by the decoded address index. In a way-halting cache, the decoded index activates only the main tag array and data arrays of ways that have *not* been predetermined by the halt tag array check to be a mismatch – predetermined mismatches effectively *halt* the access to a way's main tag array and data array. Note that way-halting does not impact the hit rate, as the hit rate is identical to that of a four-way cache – we've merely caused early terminations of accesses to ways that are pre-determined to be misses. Furthermore, through careful design, we can design the halt tag array to guarantee that the access and comparisons of the halt tag array do not extend the cache's critical path. We will show that a way-halting cache comes very close to halting three ways on a hit (and hence accessing only one full tag and data way), and to halting all ways on a miss (and hence accessing no full tag or data ways) – both approaching the ideal minimums of cache access.

Our way-halting cache uses a fully-associative memory for the four-bit-wide halt tag array. We took special care to design that memory using static circuits for the ease of implementation, in contrast to the dynamic circuits used in the content-addressable memories (CAMs) found in some modern highly associative cache architectures of embedded processors. Fortunately, address locality found in applications means that the static circuit inputs do not switch much. Since CMOS power consumption comes mostly from switching, our fully associative halt tag array memory consumes very little power.

A closely-related work is the adaptive serial-parallel highly-associative cache [5] that reduces tag access power by checking the least four significant tag bits of each tag (stored in CAMs using dynamic logic) in the first cycle, and checking the remaining bits if the first four match in the second cycle.—This design prolongs cache access time resulting 25% slowdown as reported in [5]. A way-halting cache (which was developed independently), in contrast, has no performance overhead, and uses only static logic.

II. WAY-HALTING CACHE ARCHITECTURE

A. Base architecture

Our way-halting cache architecture is shown in Figure 1. We

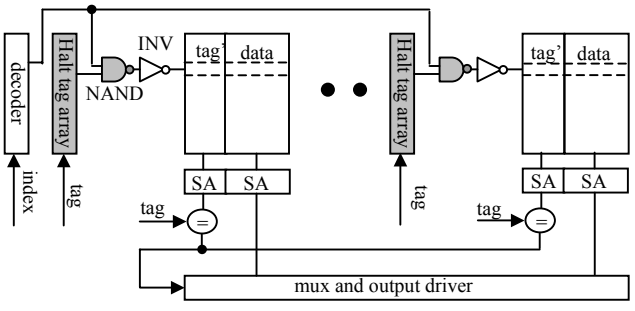

Figure 1: Way-halting four-way set-associative cache architecture. Four bits of each tag is stored in a separate halt tag array for each way. The first inverter of the word line driver is replaced by a NAND gate.

This work was supported by the National Science Foundation (CCR-0203829) and by the Semiconductor Research Corporation
Manuscript submitted: 21 Aug. 2003. Manuscript accepted: 23 Sept. 2003. Final manuscript received: 30. Sept. 2003.

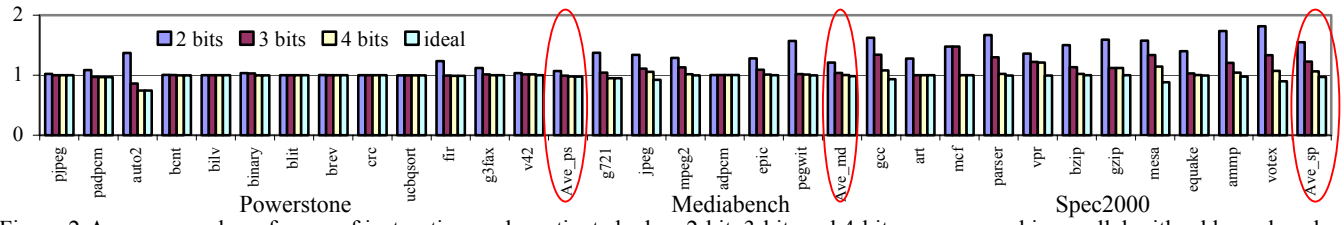

Figure 2**:**Average number of ways of instruction cache activated when 2-bit, 3-bit, and 4-bit are compared in parallel with address decoder. *Ave_ps, Ave_md, and Ave_sp* stand for average of Powerstone, Mediabench, and Spec 2000 respectively.

utilize a four-way set-associative cache as our base architecture, since four-way yields a sufficiently good hit rate for most applications. We use an 8 Kbyte total cache size and a 32-byte line size, though our approach can be applied straightforwardly to caches with other configurations. For such a cache, a memory address will be divided into 6 index bits to determine the set, 21 tag bits to determine a match, and 5 offset bits to extract the appropriate bytes from a line. The index bits from a desired address are fed into the decoder. One decoder output will become high, and is strengthened by a word line driver consisting of a pair of cascaded inverters, activating four cache lines of the one cache set. Four tag and data arrays are thus read out simultaneously through the sense amplifiers. Four comparators compare the desired address tag with the tags read from the tag array to see which way (if any) is a hit. The data of the hit way is sent to microprocessor through the mux and output driver.

B. Main idea – early detection of misses

Given a four-way set-associative cache, four tags are checked for each cache access. At most one of those tags may match, with the other three being mismatches. We observed that usually the mismatches occur in the low order tag bits because of the spatial locality of access. Therefore, if we can somehow check the low-order bits of a tag *early*, we can detect most misses early. Consequently, we can terminate the access to the full 17 bits of tag as well as to the data array banks before they consume power.

C. Basic architecture

To enable early detection of misses, we store the low-order 4 bits of each tag in a separate 4-bit-wide memory. We call this memory the *halt tag array* (see Figure 1). We call the remaining 17-bit-wide tag array, shown as *tag* in the figure, the *main tag array*. In a conventional cache, the index of the desired address is first decoded. Then the resulting decoder output line activates the read of the appropriate tags from the tag arrays. The tags are next compared with the desired tag to determine a hit or a miss. Decoding takes some time, during which we have the opportunity to check the halt tag array without increasing the tag path delay. Since the address has not been decoded yet, we do not know which tag in the halt tag array to read and compare – we therefore compare *all* the halt tags with the four bits of the desired address' tag. We accomplish this by implementing the halt tag array as a fully-associative memory, which we point out is only 4 bits wide (and 64 rows long), making such a memory feasible in terms of size and power.

In a conventional cache, the address decoder would assert a single output line high, and that line would be strengthened by the word line driver enabling reading the appropriate row from the tag and data arrays. In our way-halting cache, that output line should be ANDed by the results of the halt tag array comparison for that row. In other words, only if the low-order four bits match should the cache continue to access the main tag array and the data array; if the halt tag was a mismatch, the output line should *not* go high.

Adding an AND gate after the double inverters would lengthen the critical path. Instead, we can achieve the same logic by replacing the first inverter by a NAND gate as shown in Figure 1; the second inverter makes the total logic an AND. A NAND gate would normally be slower than an inverter. However, the first inverter of the cascaded inverters is typically small – the second inverter is instead appropriately sized larger to drive the signal. Thus, when replacing the first inverter by a NAND gate, we can increase the size of the NAND gate so that the gate's switching speed is the same as the original inverter. The identical technique of replacing the first inverter by a resized NAND gate was used in the way-concatenate cache in [13], with detailed layout and timing analysis results showing no lengthening of the critical path.

III. Designing the Halt Tag Array

The most important component in a way-halting cache is the halt tag array. It must be designed not only to be faster than the index decoder, but also to consume low enough energy so that we obtain overall energy savings. The two most important considerations in the design of the halt tag array are: (1) What bit width should the array be, and (2) how should the comparisons be implemented in the fully-associative memory.

A. Bit width of the halt tag array

We examined the impact of the halt tag array's bit width on the number of ways halted. We wanted to find the minimum number of bits that halts nearly three of the four ways per hit, or conversely stated, activates only one of the four ways per hit. We tried bit widths varying from 2 to 4. We simulated a variety of benchmarks for an 8 Kbyte, 32-byte line size cache using SimpleScalar [2]. The benchmarks included programs from Motorola's Powerstone suite [9] MediaBench [8] and eleven programs from Spec2000 [6] .We used the reference input vectors with each benchmark as program stimuli. For Spec 2000 benchmarks, we fast forwarded the first one billion instructions to warm up the caches and simulated the next 500M instructions.

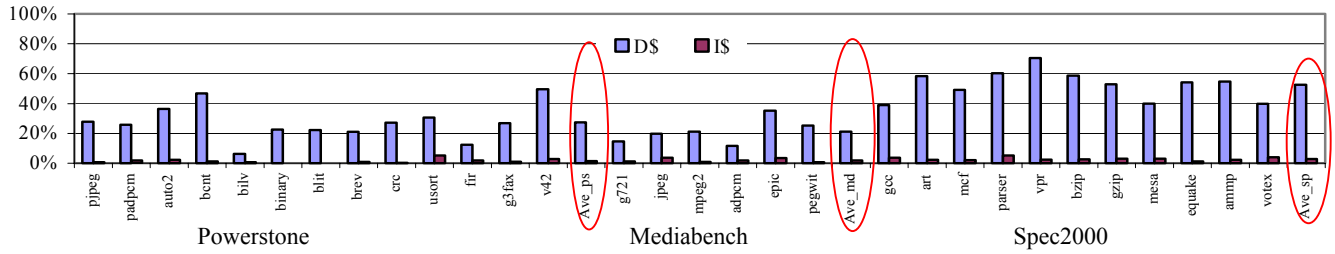

Figure 3:Tag address change frequency of data and instruction cache.

Energy will be saved if the number of ways, both tag and data, activated are reduced. We collected the average number of cache ways activated using a halt tag array width of 2, 3 and 4 bits. Results are shown in Figure 2 for instruction cache (results for data cache are not shown due to space limits) with averages for each benchmark suite circled. We see that a bit width of 4 is very close to the ideal situation of only accessing one way per hit. We also experimented with 16 Kbyte and 32 Kbyte caches and obtained similar results.

B. Halt tag array design

Each halt tag array is a 64x4 fully associative memory. If we do not design that array properly, it may consume too much energy and hence mitigate savings obtained from halting ways.

We originally designed the halt tag array using traditional 10-transistor CAM cells, utilizing dynamic circuit techniques, as found in highly-associative CAM-tag based caches. We laid out the halt tag array, as well as the rest of the cache including the main tag array and the data array SRAM, in a TSMC 0.18 micron CMOS technology obtained through MOSIS [10]. We utilized several low power SRAM design techniques such as pulse word line control to limit the bit line swing, and word line segmentation such that only one word (32 bits) is read [1] on each read access.

However, we found that designing the halt tag array as a fully-associative memory built using static circuit (SRAM-based) techniques resulted in a lower energy per access. Initially, one might be surprised at this statement, because static circuits typically consume more power than dynamic circuits. However, spatial locality works in our favor here. In particular, a static circuit only consumes power (dynamic power, that is) when its inputs change. Spatial locality implies that the halt tag address is usually the same from one access to another, in which case no dynamic power is consumed in the halt tag array. We measured the percentage of back-to-back changes in the dynamic halt tag address streams and plotted the results in Figure 3. Furthermore, even when there is a change in the address, only a few static comparators' output bits change, keeping the dynamic power low for our static circuit comparator. Another advantage of using static circuit design is that the SRAM cell and the logic tools are available off-the-shelf. Note that data cache tags change more frequently due to less spatial locality than instruction cache, but are still low.

Our halt tag array design is shown in Figure 4. One word of the array is depicted, which consists of four standard SRAM cells (two are shown), and a static comparator.

The static comparator component must execute as fast as the address decoder component to avoid lengthening the critical path. Both components have two levels of gates. We designed our XOR and NOR gates of the comparator with big enough transistors to be as fast as the address decoder. The size of one static comparator is 3 µm × 16µm. The total area overhead is less than 2% of the total cache area.

IV. EXPERIMENTS

In this section, we show the experiments result and compare the energy consumption of the way-halting cache with previously proposed low-power cache architectures, including CAM-based highly associative, direct-mapped, way prediction, phased, and pseudo-set-associative caches.

A. Energy evaluation

We compute the overall energy consumption taking into account the off-chip memory and the processor core. The energy model is given in the following equations:

1. $overall\_energy = no\_of\_hits * \underline{hit\_energy} + \underline{no\_of\_misses} * miss\_energy$

2. $miss\_energy = offchip\_access\_energy + uP\_stall\_energy + \underline{cache\_block\_fill\_energy}$

In the first equation, the *no_of_hits* and *no_of_misses* are obtained by running SimpleScalar with different cache configurations. The *hit_energy* is computed through simulation of circuits extracted from our layout of SRAM cache using Cadence [3].

Determining the *miss_energy* in the second equation is more involved. The *offchip_access_energy* value is the energy for accessing off-chip memory and the *uP_stall_energy* is the energy for the microprocessor when it

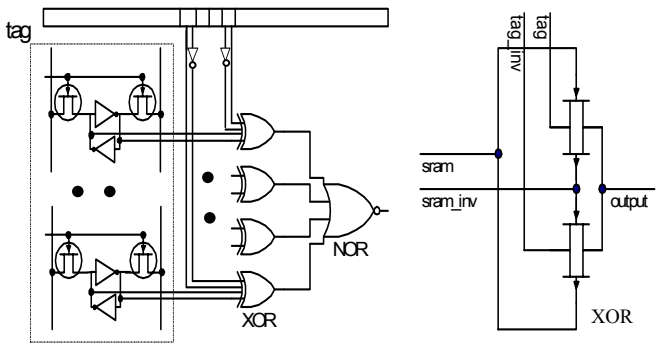

Figure 4: Design of a fully associative memory for the halt tag array, based on a static circuit only. The sixteen input static comparator is composed of four XOR gates and one NOR gate. Eight inputs come from the SRAM cells that store the halt tag, while the other eight inputs come from the desired address' least four-tag bits.

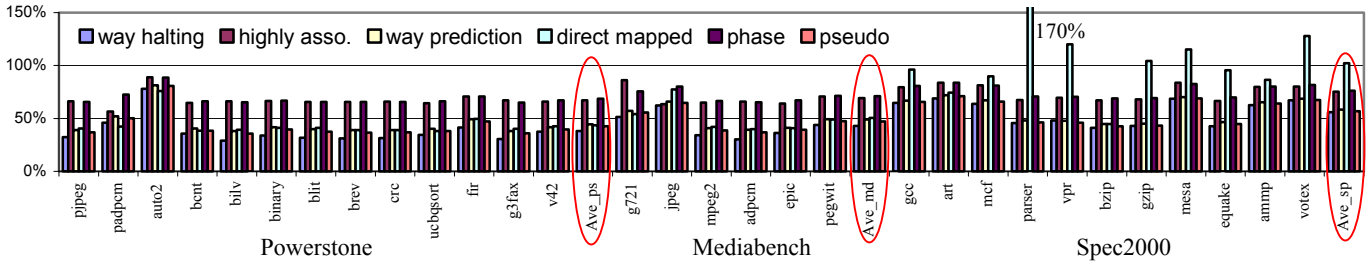

Figure 5:Energy for various cache designs, for Powerstone benchmarks, normalized to a conventional 4-way cache.

is stalled due to cache misses. The *cache_block_fill_energy* is the energy to fill the cache with a new block. The first two terms are highly dependent on the memory model and microprocessor model used in a system. Results from one real system may be entirely different from another. Therefore, we choose instead to create a "realistic" system, and then to vary the configurations to see the impacts on energy distribution. We examined all three terms in equation 2 for typical commercial memories and microprocessors. We found that *miss_energy* is 50 to 200 times the *hit_energy*. Thus, we remodeled the *miss_energy* using the following equation:

*3. miss_energy = k_miss_energy * hit_energy*

We will consider the situations where *k_miss_energy* is equal to 50 and 200 respectively.

B. Comparisons with other low power cache architectures

Figure 5 compares the energy dissipation of way-halting with CAM-based highly-associative [14], direct mapped, way predicting [11], phased [7], and pseudo-set-associative caches [4], using *k_miss_energy = 50*. The energy is normalized with respect to a conventional four-way set-associative cache equaling 100%. We see that a way-halting cache is most energy efficient. Although the energy difference compared with some of the other cache designs may seem small, bear in mind that these savings come with *no performance penalty* compared to a four-way cache.

We also generated the data for *k_miss_energy = 200*. Way-halting still dissipated the least energy on average, although highly-associative was more competitive due to the low miss rate of the high associativity – a high energy penalty (200) for off-chip access means that the lower miss rate due to high-associativity saves more energy from off-chip memory accesses than the case of just a high penalty of 50.

V. CONCLUSION

A way-halting cache is able to save, across three different benchmark suites, an average 45% to 60% of the energy of a conventional four-way set-associative cache, with only 2% area overhead, and no performance penalty – neither more cycles nor longer critical path. That energy savings is better than previous low-power cache approaches, and although the energy savings are only slightly better than some of those approaches, all those other approaches introduce performance overhead. Way-halting also saves energy over a highly-associative cache, adopting only static circuits that can be designed using standard memory compilers and tools. We designed the way-halting cache using a combination of architectural and layout methods. A key feature of our design is the use of a small fully-associative memory for the halt tag array based on a static circuit rather than a dynamic one, saving power because of the tendency of address tags to stay the same.

REFERENCES

[1] B. Amrutur and M. Horowitz, "A replica technique for word line and sense control in low-power SRAM's," IEEE Journal of Solid-State Circuits, vol. 33, no. 8, pp.1208-1218, Aug.1998

[2] D. Burger and T.M. Austin, "The SimpleScalar Tool Set, Version 2.0," Univ. of Wisconsin-Madison Computer Sciences Dept. Technical Report #1342, June 1997.

[3] Cadence, http://www.cadence.com

[4] B.Calder, D. Grunwall, and J. Emer, "Predictive Sequential Associative Cache," Int. Symp. on High Performance Computer Architecture, Feb. 1996.

[5] A. Efthymiou and J.D. Garside, "An Adaptive Serial-Parallel CAM Architecture for Low-Power Cache Blocks," Proceedings of the Int. Symp. on Low Power Electronics and Design, 2002.

[6] http://www.specbench.org/osg/cpu2000/

[7] A. Hasegawa, I.Kawasaki, K.Yamada, S.Yoshioka, S. Kawasaki, and P. Biswas, "SH3: High code density, low power," IEEE Micro, Dec. 1995.

[8] C. Lee, M. Potkonjak and W. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems," Int. Symp. on Microarchitecture, 1997.

[9] A. Malik, B. Moyer and D. Cermak, "A Low Power Unified Cache Architecture Providing Power and Performance Flexibility," Int. Symp. on Low Power Electronics and Design, June 2000.

[10] The MOSIS Service, http://www.mosis.org

[11] M. Powell, A. Agarwal, T.N. Vijaykumar, B. Falsafi, and K. Roy, "Reducing Set-Associative Cache Energy via Way-Prediction and Selective Direct-Mapping," Int. Symp. on Microarchitecture, 2001.

[12] S. Segars, "Low power design techniques for microprocessors," Int. Solid-State Circuits Conf. Tutorial, 2001

[13] C. Zhang, F. Vahid, and W. Najjar, "A Highly-Configurable Cache Architecture for Embedded Systems," Int. Symp. on Computer Architecture, 2003

[14] M. Zhang and K. Asanović, "Highly-Associative Caches for Low-Power Processors," Kool Chips Workshop, in conjunction with Int. Symp. On Microarchitecture, Dec. 2000.